Tudnivalók
az otthon kidolgozandó feladatokról

- Otthon kidolgozandó feladat megoldásának beküldése csak azok számára kötelező, akik fölvették az Assembly programozás konzultáció kurzust. Minden hallgató, aki az általa választott vagy a számára kiosztott otthon kidolgozandó feladat megoldását beküldi, a megoldás minőségétől függően minimum 0, maximum 10 pontot kap.
- A feladat választani az ETR CooSpace moduljának segítségével lehet a http://www.coosp.etr.u-szeged.hu címen **2017. március 10. 00:00-tól**. Bejelentkezni ugyanazzal a felhasználónévvel és jelszóval kell, mint amellyel az ETR-be. Bejelentkezés után az „IB676g-XX Assembly programozás gyakorlat" színteret választva a „Jelentkezés" fejléccel ellátott „Otthon kidolgozandó feladatok – jelentkezés" linkre kattintva érhető el az a felület, melynek segítségével kiválasztható a megoldani kívánt feladat. A választás határideje **2017. május 1. 23:55.**
- Egy programot maximum 2 hallgató választhat gyakorlati csoportonként. A feladatok kiosztása a választás sorrendjében történik automatikusan. Ha már mindkét hely elkelt egy adott feladatra, akkor a CooSpace már nem enged több jelentkezést.
- Az otthon kidolgozandó feladat programját Intel x86 assembly nyelven kell elkészíteni úgy, hogy a kabinetben a gyakorlaton megismert módon 32-bites környezetben Visual Studioban fordítható legyen.
- Hallgatók közötti kooperáció, külső források (internet, szakirodalom) felhasználása megengedett, de a beadott végeredményt a hallgatónak ismernie kell. Nem elfogadható az a program, amelynek a működését a készítője nem ismeri, nem tudja elmagyarázni annak főbb részeit!
- Az otthon kidolgozandó feladat beküldési határideje: **2017. május 15. 23:55**. A feladatokat szintén a CooSpace-en kell majd beadni az „Feladatok" fejléccel ellátott „Otthon kidolgozandó feladat" linken. A CooSpace lehetővé teszi a több alkalommal történő feltöltést is, de a gyakorlatvezetők mindig csak az **utoljára feltöltött** fájlt tudják figyelembe venni! A program beadására később is van lehetőség. Aki az utolsó gyakorlat előtt beadja a programot az közvetlenül a ZH után meg is védheti azt.
- A választható programok pontos működése helyenként nem teljesen van megfogalmazva. Amennyiben valami nincs konkrétan kikötve, bármilyen

technikai megoldás alkalmazható, lényeg hogy a program a várt funkcionalitást biztosítsa.

- A program kódjának tartalmaznia kell a hallgató nevét, csoportját és EHA kódját komment formájában. Szintén komment formájában tartalmaznia kell a program input/output specifikációját (azaz hogy pontosan hogyan használható) és rövid, pár soros leírását.
- Értékelés: a program elfogadásának alapfeltétele, hogy fordítani és futtatni lehessen a kabinetes környezetben Visual Studio-ban. Amennyiben ez nem teljesíthető, az otthon kidolgozandó feladat – és amennyiben a hallgató fölvette az Assembly programozás konzultációt, a kurzus teljesítése is sikertelen. **A programot az utolsó héten előre egyeztetett időpontban be kell mutatni.** A bemutatás során demonstrálni kell a programot futás közben, és röviden el kell tudni mondani annak működését, főbb részeit, szükség szerint a program megvalósításával kapcsolatos kérdésekre válaszolni.
- A program beadására később is van lehetőség. Aki az utolsó gyakorlat előtt beadja a programot az közvetlenül a ZH után meg is védheti azt. Ennek feltétele, hogy a program forráskódja fel legyen töltve a coospare-re.
- Kötelező program bemutatás utáni javítására, határidőn túli pótlására nincs lehetőség.

Leírás a programok Visual Studioban történő 32-bites fordításáról:

- VS 2010 alatt: http://www.kipirvine.com/asm/gettingStartedVS2010/index.htm
- VS 2012 alatt: http://www.kipirvine.com/asm/gettingStartedVS2012/index.htm

Otthoni kidolgozásra választható feladatok listája

1. **Ceasar-kódolás** *(http://hu.wikipedia.org/wiki/Caesar-rejtjel)* **(10 pont)**
   - Input: egy karakter és egy karaktersorozat bekérése a billentyűzetről
   - Output: az inputon kapott karaktersorozat kódolt vagy dekódolt formában kiírva a képernyőre
   - Feladat részletesen: Az első karakter azt mondja meg, hogy kódolni vagy dekódolni szeretnénk az input szöveget (pl.: 1-kódolás, 2-dekódolás). Kódolás esetén végig haladunk a szövegen és minden egyes betűt az ábécében egy tőle meghatározott távolságra lévő betűvel kell helyettesíteni. Így például, ha mondjuk az eltolódás 3, az **angol** ábécében az A-t a D-vel, a B-t az E-vel stb. kell helyettesíteni. Ezt az eltolást standard módon előre megadhatjuk a programkódban. Dekódolás esetén hasonló a helyzet, csak az eltolást visszafelé végezzük. Ügyelni kell arra, hogy például, ha az eltolás 3, az angol ábécé utolsó 3 betűjét, az ábécé első 3 betűjével helyettesítsük, és fordítva, ne fussunk ki az ábécéből. (X<->A, Y<->B, Z<->C).
2. **Egyábécés helyettesítő kódolás kulcs alapján (10 pont)**
   - Input: egy karaktersorozat bekérése a billentyűzetről
   - Output: az inputon kapott karaktersorozat kódolt formában kiírva a képernyőre
   - Feladat részletesen: Adott egy 26 karakter hosszú kulcs, mely az angol ábécé betűinek permutációja. Az egyábécés helyettesítő kódolás a kulcs alapján úgy történik, hogy az angol ábécé betűit a kulcs adott pozícióján lévő betűjével helyettesítjük. Tehát az angol ábécé betűit (,,abcdefghijklmnopqrstuvwxyz") rendre helyettesítjük a kulcsban szereplő (pl. ,,dkvqfibjwpescxhtmyauolrgzn") betűkkel, vagyis, az 'a' betűket 'd' betűkre, a 'b' betűket 'k' betűkre, a 'c' betűket 'v' betűkre, és így tovább lecseréljük. A helyettesítő kulcsot megadhatjuk előre a kódban. Az egyértelműség kedvéért használjuk az itt megadott helyettesítő kulcsot. Elég csak a kisbetűket lekezelni.
3. **Számolás megvalósítása nagy számokkal – Összeadó gép (10 pont)**
   - Input: a felhasználó billentyűzetről maximum 20 jegyű egész számokat és műveleteket ír be. Sorban egy szám, majd egy művelet. A műveletek: + (összeadás) és – (kivonás), Enter jelzi a formula végét. A program csak az éppen következő inputnak megfelelő karaktereket engedje beírni (vagy számjegyet, vagy műveleti jelet).

- Output: a beírt számokkal a műveleteket elvégzi, és az eredményt kiírja a képernyőre. A kapott eredménnyel számoljunk tovább egészen addig, amíg a felhasználó ki nem lép.
- Megjegyzés: Legegyszerűbb, ha úgy dolgozunk, mintha papíron végeznénk a számolást.

4. **Számolás megvalósítása nagy számokkal (input, kiírás, megvalósítás) – Szorzás (10 pont)**
   - Input: két, maximum 10 jegyű egész számot olvassunk be billentyűzetről és írjuk is ki a képernyőre. A számok végét Enter leütésével jelezheti a felhasználó, de 10 számjegy után automatikusan lépjünk a következő számra.
   - Output: ha megvan a két szám, szorozzuk őket össze és írjuk ki az eredményt a képernyőre.
   - Megjegyzés: Legegyszerűbb, ha úgy dolgozunk, mintha papíron végeznénk a számolást.
5. **Anagramma ellenőrzés (10 pont)**
   - Input: olvassunk be a billentyűzetről két szót. A szavak végét szóköz jelezze.
   - Output: írjuk ki a két szó alá, hogy anagramma, ha az egyik szó a másik szó betűinek permutációjával előállítható, különben azt írjuk ki, hogy nem anagramma.
6. **Betű hisztogram (10 pont)**
   - Számoljuk meg, hogy az input karaktersorozatban melyik betű hányszor fordul elő.
   - Input: A klaviatúráról beolvasott legfeljebb 80 karakter hosszú karaktersorozat.
   - Output: a képernyőre kiírt táblázat tartalmazza az előforduló betűket és az előfordulások számát.
7. **Szó ismétlések száma (12 pont)**
   - Az input szavait szóközök választják el egymástól. Meg kell számolni, hogy hányszor fordul elő az, hogy egy szó megismétlődik az inputban.
   - Input: A klaviatúráról beolvasott legfeljebb 80 karakter hosszú karakter sorozat.
   - Output: a szó ismétlések száma.
8. **Hamming kódolás (10 pont)**
   - Olvassunk be a billentyűzetről egy nyolcjegyű hexadecimális számot, és írjuk ki a páros paritású Hamming kódolását binárisan!

9. **Bit hibajavítás (10 pont)**
   - Olvassunk be a billentyűzetről egy 32 bites bináris szám páros paritású Hamming kódját (38 bites szám)! Állapítsuk meg, hogy hibás-e! Ha hibás, akkor írjuk alá a helyes számot, ha helyes, akkor írjuk ki, hogy helyes!
10. **Prímszámkereső (10 pont)**
   - Keressük meg és írjuk ki a képernyőre az összes prímszámot 1 és 100 000 között.
11. **Tökéletes számok (10 pont)**
   - Keressük meg és írjuk ki a képernyőre az összes szóban (16 biten) elférő tökéletes számot!
12. **Barátságos számok (10 pont)**
   - Keressük meg, és írjuk ki a képernyőre az összes szóban (16 biten) elférő barátságos számpárt!
13. **Hanoi tornyai (12 pont)**
   - Input: A klaviatúráról beolvasott három egész szám, az első 1 és 10, a többi 1 és 3 közötti.
   - Output: A Hanoi tornyai játék képernyőre kiírt megoldása feltételezve, hogy az első szám a torony magasságát jelöli, a másik kettő pedig a kezdő és a cél pozíciókat.
   - Segítség: A rekurzió leírása az előadás diasor végén található.
14. **Newtoni gyökvonó algoritmus (10 pont)**
   - Feladat:

     http://www.tankonyvtar.hu/hu/tartalom/tamop412A/2010-0012_szemleletes_analizis/ch04s07.html

     A Newton-i gyökvonó algoritmus megírása.
   - Input: A szám amiből gyököt kell vonni, ezt olvassuk be billentyűzetről.

     X0-t pedig seedeljük meg tetszőlegesen. Ez lehet pl const. 1, egy random szám, vagy az input valamilyen függvénye, pl fele vagy negyede az eredeti számnak.
   - Output: Legalább a szám gyöke, és lehetőség legyen egyszerűen, pl. 1 kikommentelt sor visszatevésével a részeredmények kiíratására is.
15. **Gyorsrendezés (számok) (13 pont)**
   - Input: egy maximum 10 elemű, 32 bites előjeles egész számokat tartalmazó tömb az adatszegmensben.
   - Output: készítsük el a tömb rendezett változatát a gyorsrendezés algoritmusa szerint. A rendezést ne helyben végezzük el, hanem

készítsünk a tömbről másolatot előtte. Az eredményt írjuk ki a képernyőre, vesszővel elválasztva.

16. **Buborékrendezés (sztringek) (11 pont)**
    - Input: egy 5 elemű, max. 10 karakter hosszú sztringre mutató pointerből álló tömb az adatszegmensben + a hozzá tartozó sztringek. A sztringek hosszúsága változó, és 0 értékű byte jelzi a végüket.
    - Output: készítsük el a tömb rendezett változatát a buborékrendezés algoritmusa szerint. A rendezést helyben végezzük el. Az eredményt írjuk ki a képernyőre.
17. **Beszúró rendezés (12 pont)**
    - Feladat: A beszúró rendezés megírása.
    - Input: A rendezendő tömb az adatszegmensben
    - Output: A rendezett tömb.
18. **Törtszámok megvalósítása (10 pont)**
    - Valósítsuk meg a 4 alapműveletet valódi törtszámoláshoz. A törteket számláló/nevező alakban tároljuk, egyszerűsítve minden esetben. Egész számokat is törtként x/1 alakban kezelhetjük. Elég, ha számláló és a nevező belefér egy-egy regiszterbe.
    - A műveletek bemutatásához készítsünk kis programot, ami 1-1 példaszámolást végez mindegyik műveletre. Az eredményt számláló/nevező alakban kell kiírni.
19. **Számátváltó 2-16 rendszerek között, oda-vissza (10 pont)**
    - Készítsünk számrendszerek közötti átváltó programot.
    - Input: számrendszer típusa (2-16 közötti szám), majd egy szám beírása billentyűzetről és végül újból egy számrendszer típusa. Az adatok beírását Enter zárja.
    - Output: az elsőnek megadott számrendszerből váltsuk át a beírt számot a másodikba. Ügyeljünk arra, hogy csak az elsőnek beírt számrendszernek megfelelő számjegyeket engedjünk beírni! Az eredményt írjuk ki a képernyőre.
20. **Mértani sorozat (10 pont)**
    - Állapítsuk meg, hogy egy számsorozat mértani sorozatot alkot-e.
    - Input: a billentyűzetről beolvasott előjel nélküli számok sorozata.
    - Output: "Mértani sorozat", ha a számok mértani sorozatot alkotnak. "Nem mértani sorozat" egyébként.

21. Rész-string keresés (12 pont)

- Input: A klaviatúráról beolvasott két, legfeljebb 80 karakter hosszú karaktersorozat.
- Output: „Nincs egyezés.” kiírása a képernyőre, ha az első karaktersorozat nem tartalmazza a másodikat, különben egy egész szám, az a pozíció ahonnan kezdve az első sorozatban szerepel a második.

22. Növekvő részsorozat hossza (10 pont)

- Határozzuk meg egy számsorozat leghosszabb növekvő részsorozatának a hosszát
- Input: számsorozat.
- Output: a leghosszabb növekvő részsorozat hossza.

23. Mátrixszorzás (13 pont)

- Feladat: A kapott legfeljebb 20x20-as mátrixok összeszorzása.

  Input:

  - Valahány mátrix az adatszegmensben mérettel együtt.
  - Vagy konzolról beolvasva plusz pontért, de ekkor nem használhatunk fel több memóriát, mint 3 db 20x20 tömb és 3x2 változó a mátrix méretek tártolására.

    A konzolos input a következő formátumot kövesse:

    x

    m1 n1

    a11 a12 a13

    a21 a22 a23

    a31 a32 a33

    m2 n2

    b11 b12

    b21 b22

    b31 b32

    Ahol x a kapott mátrixok száma, m a sorok száma, n az oszlopok száma, aij pedig a mátrixok elemei. A mátrixokban elég, ha csak egész elemekkel foglalkozunk.

- Output: A végeredmény.

24. Akasztófajáték (12 pont)

- Szókitalálós játék előre megadott szókészlettel. A szavakat az adat szegmensben tároljuk, változó hosszúságú 0 értékű byte-ra végződő sztringeket használva. Induláskor a gép sorsolja ki az egyik szót és írjon ki annyi „-” jelet a képernyőre ahány betűből a szó áll. A játékos betűk lenyomásával tippelhet. Ha a szóban van olyan betű, akkor írjuk ki a megfelelő helyre a „-” jel helyére. Ha nincs, akkor növeljük a rossz tippek számát. Ügyeljünk arra, hogy csak betűket vegyünk tippnek, és a kis/nagybetűket ne különböztessük meg! Adott számú rossz tipp után, vagy amennyiben kitalálták a szót, a program írjon ki ennek megfelelő üzenetet és lépjen ki.
- +2 pontért akasztófa, ember megjelenítéssel.

25. Snake (15 pont)

- Feladat: http://patorjk.com/games/snake/

  Egy hasonló snake program megírása assembly-ben.
  Mivel multi-threading nincs/nem tanítjuk, a programot úgy érdemes megírni, hogy az irányokat az Irvine non-blocking ReadKey funkciójával olvassuk be, a kígyó mozgását, pedig a ciklusok számához, az órajel függvényében arányosítjuk.

- Input: Képernyő mérete, és a processzor órajele, (akár hard-coded az adat szegmensben)

- Output: A játék megjelenítése.

  Plusz pont a grafikus megjelenítésért Mode 13-ban **(15 pont)**.

26. Amőba (15 pont)

- Feladat: Egy két személyes amőba program írása végállapot ellen
- Input: A lépések
- Output: A játék állása
- Pluszpont: Gépi játékos írása.

27. Sakk (15 pont – max 15 pont)

- Feladat: Megírni egy kétszemélyes sakk programot assembly-ben, karakteresen, vagy grafikusan (megrajzolva) plusz pontért.

  A sakkban mindig a fehér kezd, a vezér (queen) a saját színén áll. A program ellenőrizze minden lépés helyességét.

  A lépések megadására két módot javaslunk,

- koordináta párokkal, pl (C7, C5)
- nyilakkal és enterrel lehessen kiválasztani, hogy melyik bábuval szeretnénk lépni, és szintén így, hogy hova.

A 2) módszernél plusz pontot tudunk felajánlani, ha a program kijelzi, hogy hova léphet a kiválasztott bábú, és/vagy a nyilakkal csak a szabályos helyekre enged minket navigálni.

- Input: A lépések.
- Output: A tábla aktuális állása, minden lépés után.
- Plusz pontokért:
  - Sáncolás szabályos lehetősége: (+2 pont)

    https://hu.wikipedia.org/wiki/S%C3%A1ncol%C3%A1s/
  - Matt ellenőrzés (+3 pont)
  - Döntetlenség ellenőrzés (elfogy minden a 2 királyon kívül) (+3 pont)
  - Fel lehessen adni. (+1 pont)

28. Életjáték (15 pont)

Feladat: https://hu.wikipedia.org/wiki/%C3%89letj%C3%A1t%C3%A9k

A feladat egy ilyen program implementálása assemblyben.

Input: A kezdőkonfiguráció billentyűzetről.

Output: A játék aktuális állása.

29. Space Invaders (15 pont)

- Feladat: http://www.pacxon4u.com/space-invaders/

  Egy Space Invaders implementáció karakteresen (13 pont), vagy grafikusan (15 pont).

  Karakteres megvalósítás esetén, launch platformnak elég egy 3 karakterből álló valamilyen karakter, mozgó enemy-knek valamilyen karakter. A mozgatást az Irvine lib ReadKey utasításával olvassuk be.
- Input: Mozgás, lövés. Legyen az adatszegmensben a cpu-hoz arányosított frissítési ráta, és a képernyő mérete.
- Output: A játék megjelenítése.

30. Labirintus (15 pont)

- Feladat: Az adat szegmensben adott egy labirintus, amiben definiálva van egy kezdő és egy végpont.

  A cél egy ágens megírása, ami eljut a kezdő pontból a végpontig.
- Input: Labirintus egy tömbben, az adatszegmensben
- Output: Az ágens mozgása, és a labirintus.